Blockchain Lab
Experiment-3

Sanket More

D20A 35

AIM: Create a Cryptocurrency using Python and perform mining in the Blockchain created.

THEORY:-

1. Blockchain Overview

A blockchain is a distributed, decentralized, and cryptographically secured digital ledger used to record transactions in a chronological and immutable manner. Unlike traditional centralized databases that rely on a trusted third party, blockchain operates on a peer-to-peer network where trust is established through cryptographic verification and consensus protocols.

Each block in a blockchain contains:

- A set of verified transactions
- A timestamp indicating the time of block creation
- The hash of the previous block, which links blocks together
- A nonce used in the mining process
- The block's own cryptographic hash

The linking of blocks using cryptographic hashes creates a chain structure. Any attempt to modify data in a block changes its hash, which invalidates all subsequent blocks. This feature ensures immutability, data integrity, and tamper resistance.

Because copies of the blockchain are stored across multiple nodes, transparency and fault tolerance are also achieved.

2. Mining

Mining is the process by which new blocks are created, validated, and added to the blockchain. It plays a crucial role in maintaining the security and reliability of the blockchain network.

Mining performs two major functions:

1. Verification and validation of transactions
2. Securing the blockchain against malicious attacks

Mining is based on the **Proof-of-Work (PoW)** algorithm. In PoW, miners compete to solve a computationally complex mathematical puzzle. This involves repeatedly modifying a value called a *nonce* and recalculating the block's hash until it satisfies the network's difficulty criteria (such as a specific number of leading zeroes).

The difficulty level dynamically adjusts to control the rate at which blocks are mined. Once a miner successfully finds a valid hash, the block is broadcast to all other nodes for verification. Due to the high computational cost involved, PoW makes attacks such as double spending and data manipulation extremely difficult.

3. Multi-Node Blockchain Network

A multi-node blockchain network consists of several independent nodes participating in transaction validation and block propagation. In this experiment, three nodes are deployed on different ports (5001, 5002, and 5003) to simulate a real-world decentralized environment.

Each node in the network:

- Operates autonomously
- Maintains a complete copy of the blockchain
- Communicates with peer nodes using a peer-to-peer protocol

This architecture eliminates the need for a central server and ensures decentralization, data redundancy, and high availability. Even if one node fails or behaves maliciously, the network continues to function correctly. This setup clearly demonstrates how blockchain systems maintain reliability and consistency across distributed systems.

4. Consensus Mechanism

A consensus mechanism is a protocol that enables all nodes in a blockchain network to agree on a single, consistent version of the ledger. Since nodes operate independently and blocks may be mined simultaneously, temporary inconsistencies or forks can occur.

In this system, the **Longest Chain Rule** is used as the consensus mechanism. According to this rule:

- The blockchain with the maximum number of blocks is considered valid
- The longest chain represents the greatest cumulative computational effort
- Nodes discard shorter chains and adopt the longest valid chain

This rule ensures eventual consistency and prevents permanent divergence in the blockchain. Consensus mechanisms are essential for maintaining trust, preventing double spending, and ensuring the integrity of distributed ledgers.

5. Transactions and Mining Reward

A transaction is a digitally signed record representing the transfer of value between participants in a blockchain network. Each transaction contains:

- Sender's address
- Receiver's address
- Transaction amount

Transactions are verified using cryptographic techniques and temporarily stored in a transaction pool until they are included in a block.

To incentivize miners, the system introduces a mining reward mechanism. When a block is successfully mined, a special transaction called a **coinbase transaction** is created. This transaction generates new cryptocurrency and awards it to the miner. The reward mechanism encourages honest participation and provides economic security to the network.

6. Chain Replacement

Chain replacement is a synchronization mechanism that ensures all nodes in the blockchain network maintain a consistent and up-to-date ledger. Due to network latency or simultaneous mining, nodes may temporarily have different versions of the blockchain.

When the `/replace_chain` endpoint is invoked:

1. A node requests blockchain data from all peer nodes
2. The received chains are validated for correctness and integrity
3. The node replaces its own blockchain if a longer and valid chain is found

This mechanism resolves conflicts, eliminates forks, and ensures that all nodes converge to a single authoritative blockchain. Chain replacement is essential for maintaining decentralization without central coordination.

CODE:-

```
import datetime
import hashlib
import json
from flask import Flask, jsonify, request
import requests
from uuid import uuid4
from urllib.parse import urlparse

# ---------------- Blockchain Class ----------------

class Blockchain:

    def __init__(self):
        self.chain = []
        self.transactions = []
        self.create_block(proof=1, previous_hash='0')
        self.nodes = set()
    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': str(datetime.datetime.now()),
            'proof': proof,
            'previous_hash': previous_hash,
            'transactions': self.transactions
        }
        self.transactions = []     # clear after mining
        self.chain.append(block)
        return block

    def get_previous_block(self):
        return self.chain[-1]

    def proof_of_work(self, previous_proof):
        new_proof = 1
        while True:
            hash_operation = hashlib.sha256(str(new_proof**2 -
previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                return new_proof
            new_proof += 1

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        previous_block = chain[0]
        index = 1

        while index < len(chain):
            block = chain[index]
            if block['previous_hash'] != self.hash(previous_block):
                return False

            proof = block['proof']
            previous_proof = previous_block['proof']
            hash_operation = hashlib.sha256(str(proof**2 -
previous_proof**2).encode()).hexdigest()

            if hash_operation[:4] != '0000':
                return False

            previous_block = block
            index += 1

        return True

    def add_transaction(self, sender, receiver, amount):
        self.transactions.append({
            'sender': sender,
            'receiver': receiver,
            'amount': amount
        })
        return self.get_previous_block()['index'] + 1

    def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)

    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)

        for node in network:
            response = requests.get(f'http://{node}/get_chain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                if length > max_length and self.is_chain_valid(chain):
                    max_length = length
                    longest_chain = chain

        if longest_chain:
            self.chain = longest_chain
            return True

        return False

# ---------------- Flask App ----------------

app = Flask(__name__)
node_address = str(uuid4()).replace('-', '')
blockchain = Blockchain()

@app.route('/mine_block', methods=['GET'])
def mine_block():
    previous_block = blockchain.get_previous_block()
    proof = blockchain.proof_of_work(previous_block['proof'])
    previous_hash = blockchain.hash(previous_block)

    blockchain.add_transaction(node_address, 'Richard', 1)
    block = blockchain.create_block(proof, previous_hash)

    return jsonify(block), 200

@app.route('/get_chain', methods=['GET'])
def get_chain():
    return jsonify({'chain': blockchain.chain, 'length':
len(blockchain.chain)}), 200

@app.route('/is_valid', methods=['GET'])
def is_valid():
    return jsonify({'valid':
blockchain.is_chain_valid(blockchain.chain)}), 200

@app.route('/add_transaction', methods=['POST'])
def add_transaction():
    data = request.get_json()
    required = ['sender', 'receiver', 'amount']

    if not all(k in data for k in required):
        return "Missing fields", 400
    index = blockchain.add_transaction(data['sender'], data['receiver'],
data['amount'])
    return jsonify({'message': f'Transaction added to block {index}'}),
201

@app.route('/connect_node', methods=['POST'])
def connect_node():
    data = request.get_json()
    nodes = data.get('nodes')

    if nodes is None:
        return "No nodes", 400

    for node in nodes:
        blockchain.add_node(node)

    return jsonify({'nodes': list(blockchain.nodes)}), 201

@app.route('/replace_chain', methods=['GET'])
def replace_chain():
    replaced = blockchain.replace_chain()
    return jsonify({'replaced': replaced, 'chain': blockchain.chain}), 200

# ---------------- Run Node ----------------

app.run(host='0.0.0.0', port=5001)
```

OUTPUT:-

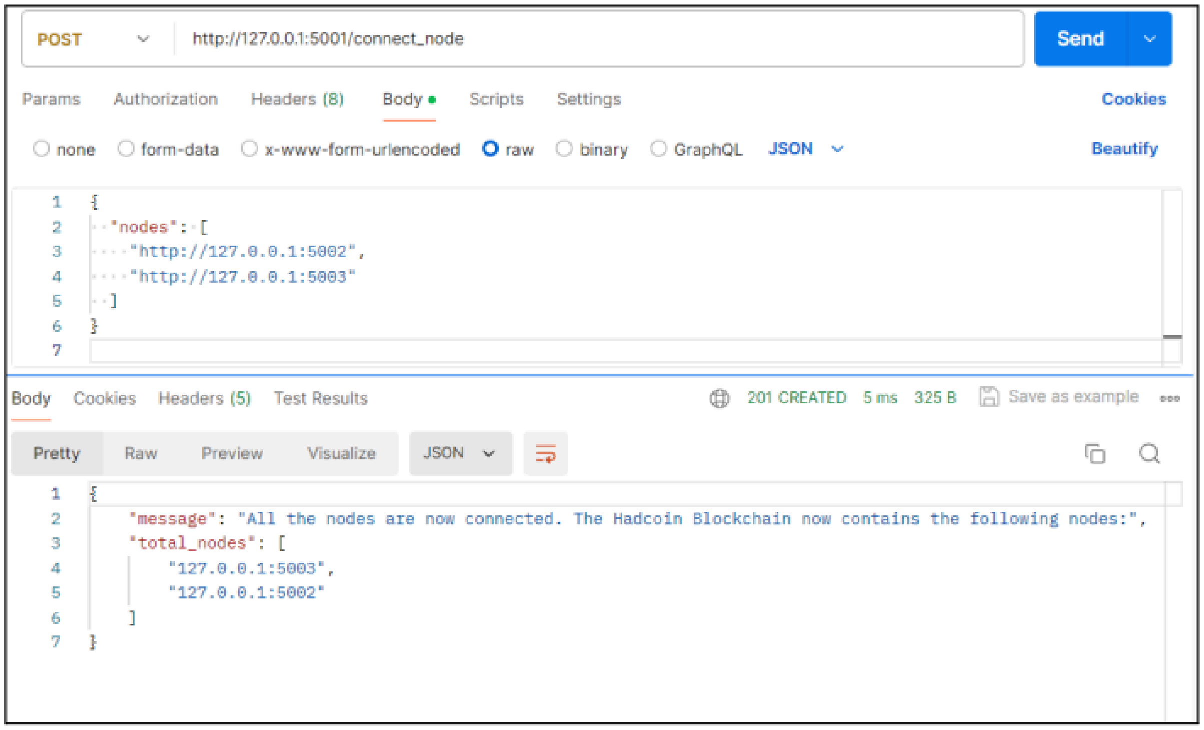
1. Connect Nodes (POST)

URL (from any node):

```
POST http://127.0.0.1:5001/connect_node
```

Body → raw → JSON

```
{
  "nodes": [
    "http://127.0.0.1:5002",
    "http://127.0.0.1:5003"
  ]
}
```



POST http://127.0.0.1:5001/connect_node

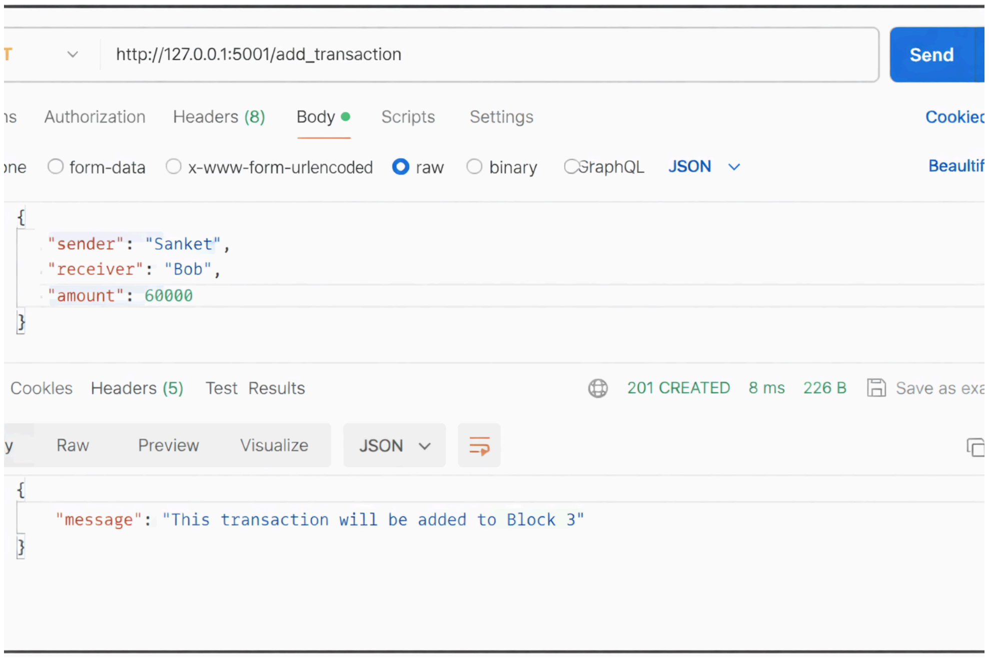
Body: raw, JSON

```
{
  "nodes": [
    "http://127.0.0.1:5002",
    "http://127.0.0.1:5003"
  ]
}
```

201 CREATED 5 ms 325 B

```
{
    "message": "All the nodes are now connected. The Hadcoin Blockchain now contains the following nodes:",
    "total_nodes": [
        "127.0.0.1:5003",
        "127.0.0.1:5002"
    ]
}
```

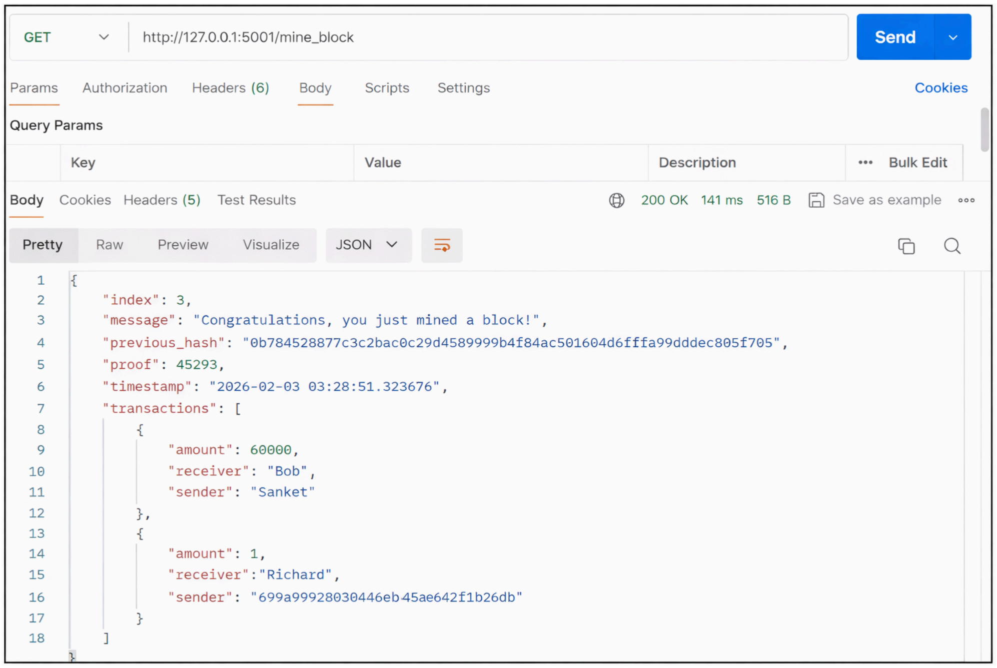
2. Add Transaction (POST)
POST http://127.0.0.1:5001/add_transaction



```
{
    "sender": "Sanket",
    "receiver": "Bob",
    "amount": 60000
}
```

```
{
    "message": "This transaction will be added to Block 3"
}
```

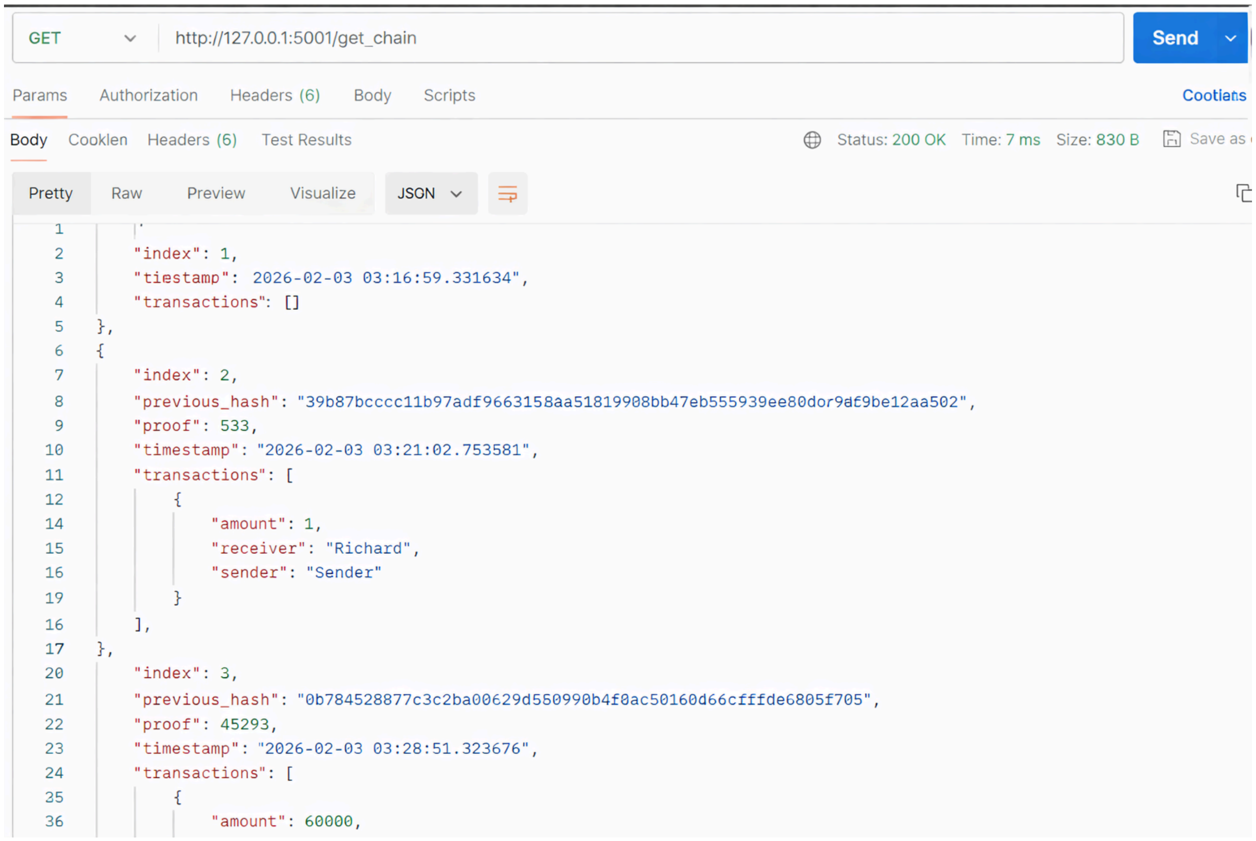
3. Mine Block (GET)
GET http://127.0.0.1:5001/mine_block



```
{
    "index": 3,
    "message": "Congratulations, you just mined a block!",
    "previous_hash": "0b784528877c3c2bac0c29d4589999b4f84ac501604d6fffa99dddec805f705",
    "proof": 45293,
    "timestamp": "2026-02-03 03:28:51.323676",
    "transactions": [
        {
            "amount": 60000,
            "receiver": "Bob",
            "sender": "Sanket"
        },
        {
            "amount": 1,
            "receiver":"Richard",
            "sender": "699a99928030446eb45ae642f1b26db"
        }
    ]
}
```

4. Get Blockchain (GET)
GET http://127.0.0.1:5001/get_chain
You'll see:

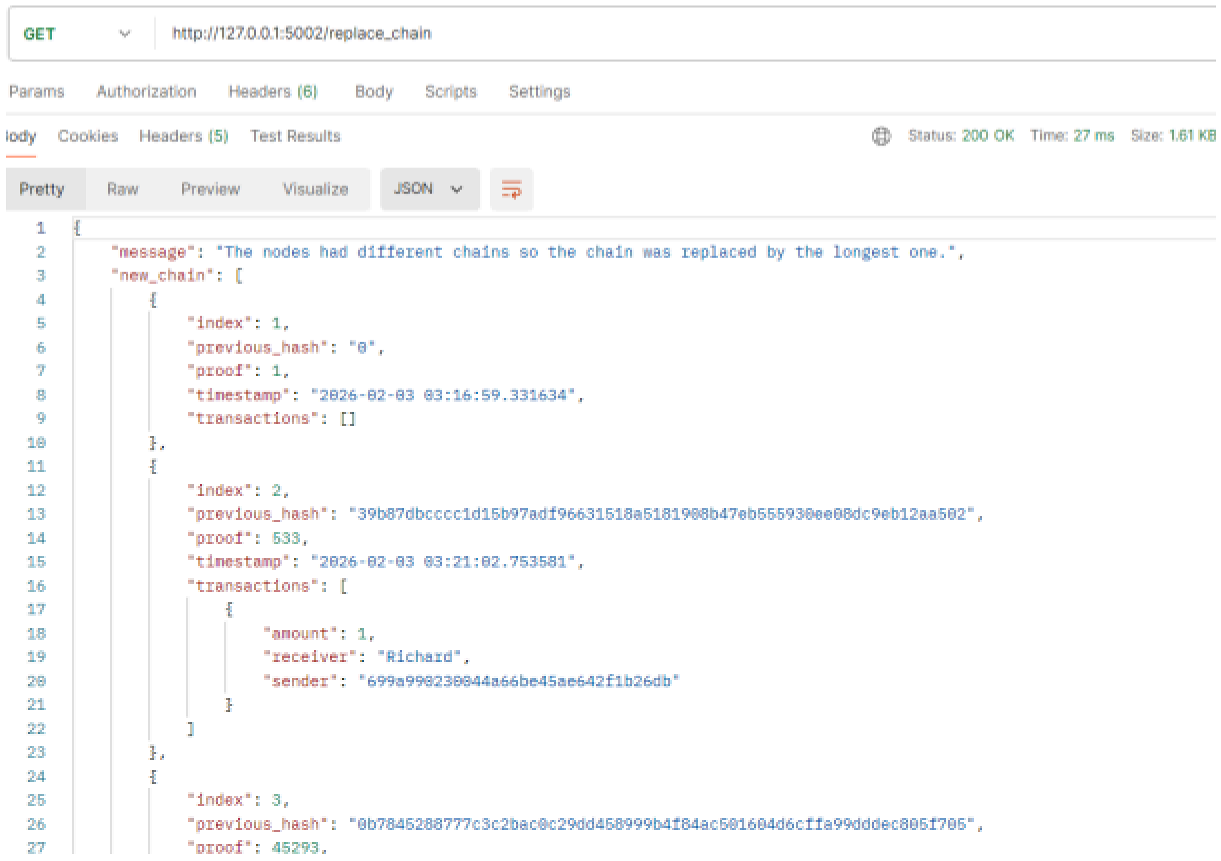
- **Transactions inside blocks**
- **transactions: [] for new pending list**



GET http://127.0.0.1:5001/get_chain

```
    '
    "index": 1,
    "tiestamp": 2026-02-03 03:16:59.331634",
    "transactions": []
},
{
    "index": 2,
    "previous_hash": "39b87bcccc11b97adf9663158aa51819908bb47eb555939ee80dor9df9be12aa502",
    "proof": 533,
    "timestamp": "2026-02-03 03:21:02.753581",
    "transactions": [
        {
            "amount": 1,
            "receiver": "Richard",
            "sender": "Sender"
        }
    ],
},
    "index": 3,
    "previous_hash": "0b784528877c3c2ba00629d550990b4f8ac50160d66cfffde6805f705",
    "proof": 45293,
    "timestamp": "2026-02-03 03:28:51.323676",
    "transactions": [
        {
            "amount": 60000,
```

5. Replace Chain (Consensus)
GET http://127.0.0.1:5002/replace_chain
Shorter chains get replaced by the longest valid one.


GET
Params Authorization Headers (6) Body Scripts Settings
Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON

http://127.0.0.1:5002/replace_chain

Status: 200 OK Time: 27 ms Size: 1.61 KB

```
{
    "message": "The nodes had different chains so the chain was replaced by the longest one.",
    "new_chain": [
        {
            "index": 1,
            "previous_hash": "0",
            "proof": 1,
            "timestamp": "2026-02-03 03:16:59.331634",
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "39b87dbcccc1d15b97adf96631518a5181908b47eb555930ee08dc9eb12aa502",
            "proof": 533,
            "timestamp": "2026-02-03 03:21:02.753581",
            "transactions": [
                {
                    "amount": 1,
                    "receiver": "Richard",
                    "sender": "699a990230044a66be45ae642f1b26db"
                }
            ]
        },
        {
            "index": 3,
            "previous_hash": "0b7845288777c3c2bac0c29dd458999b4f84ac501604d6cffa99dddec805f705",
            "proof": 45293,
```

CONCLUSION:-

In this experiment, a basic cryptocurrency system was successfully implemented using Python by incorporating fundamental blockchain principles such as block creation, cryptographic hashing, Proof-of-Work mining, transaction processing, and decentralization. A multi-node blockchain network was developed using Flask, enabling independent nodes to communicate with each other while maintaining synchronized copies of the distributed ledger.

The mining process demonstrated how new blocks are generated through computational effort, validated across the network, and rewarded with cryptocurrency, emphasizing the role of Proof-of-Work in ensuring security and integrity. Transaction management was also effectively implemented, where validated transactions were grouped into blocks and permanently recorded on the blockchain.

Overall, this experiment provided valuable hands-on experience with blockchain architecture and operations. It enhanced understanding of decentralized systems, consensus mechanisms, and mining processes, offering practical insight into how real-world cryptocurrencies function in secure, distributed environments.